03

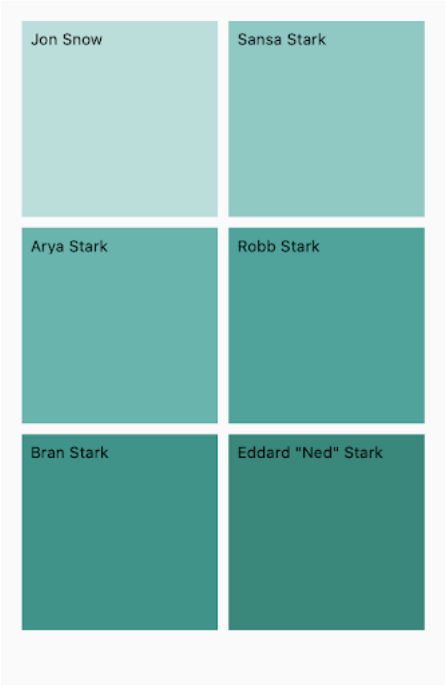
Para saber mais: Criação de layout com GridView

Existem algumas formas de utilizarmos o elemento GridView. Uma das formas é utilizar, no lugar do método `.builder`, o método `.count`.

O método `.count` também gera uma grid, igual a que temos no layout do nosso projeto. A principal diferença entre o método `.count` e o método `.builder` é que, no método `.count` precisamos já ter definido previamente quantos elementos temos dentro do nosso GridView.

O código abaixo exemplifica o uso do `GridView.count`:

```
GridView.count(
    padding: const EdgeInsets.all(20),
    crossAxisSpacing: 10,
    mainAxisSpacing: 10,
    crossAxisCount: 2,
    children: <Widget>[
        Container(
            padding: const EdgeInsets.all(8),
            child: const Text("Jon Snow"),
            color: Colors.teal[100],
        ),
        Container(
            padding: const EdgeInsets.all(8),
            child: const Text('Sansa Stark'),
            color: Colors.teal[200],
        ),
        Container(
            padding: const EdgeInsets.all(8),
            child: const Text('Arya Stark'),
            color: Colors.teal[300],
        ),
        Container(
            padding: const EdgeInsets.all(8),
            child: const Text('Robb Stark'),
            color: Colors.teal[400],
        ),
        Container(
            padding: const EdgeInsets.all(8),
            child: const Text('Bran Stark'),
            color: Colors.teal[500],
        ),
        Container(
            padding: const EdgeInsets.all(8),
            child: const Text('Eddard "Ned" Stark'),
            color: Colors.teal[600],
        ),
    ],
)
```



Podemos usar o método `.count` quando já temos as informações que queremos exibir dentro da nossa aplicação, ou seja, não estamos esperando que essas informações venham de um servidor.

O método `.count` é bem parecido com outros elementos como o `ListView` e `Row`, pois aceita uma lista de widgets como filhos. Outra diferença é que no método `.count` temos propriedade facilitadoras para construção de layout, como é o caso da `crossAxisCount`, que nos ajuda a definir quantas colunas queremos sem a necessidade de usar um `gridDelegade`.

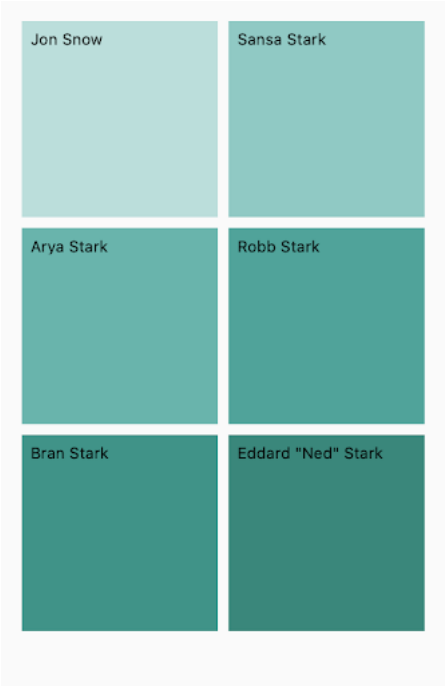
No caso de uma loja virtual, essas informações vêm por meio de conexões com um servidor e não temos como saber ao certo quantos itens podemos ter para exibir, nesse caso utilizamos o método `.builder` que é uma forma dinâmica de construir uma GridView.

Exemplo de código utilizando o GridView.builder:

```
List<String> nomes = [
    'Jon Snow',
    'Sansa Stark',
    'Arya Stark',
    'Robb Stark',
    'Bran Stark',
    'Eddard "Ned" Stark'
];

return Scaffold(
    body: GridView.builder(
        padding: const EdgeInsets.all(20),
        gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(crossAxisCount: 2),
        itemCount: nomes.length,
        itemBuilder:(BuildContext context, int index) {
            return Container(
                padding: EdgeInsets.all(8),
                margin: EdgeInsets.all(8),
                child: Text(nomes[index]),
                color: Colors.teal[(index + 1) * 100]
            );
```

```
            },
        )
    );
```



Quando temos informações que vêm direto de um servidor ou alguma outra fonte dinâmica e queremos dispor essas informações como um *grid* (grade), devemos usar o método `.builder` no GridView.

A propriedade `itemBuilder` dentro do método `.builder` serve como um laço de repetição e nos provém um número inteiro, que é o índice do laço. Isso nos permite acessar item por item das informações que precisamos exibir, desde que elas estejam organizada com uma estrutura de dados que pode ser acessado usando um índice, como é o caso das listas.

Também devemos informar qual é número de elementos que o `GridView` deve iterar, ou se o elemento cria uma lista infinita de elementos. Informamos esse valor em uma outra propriedade do `GridView.builder`, a `itemCount`. Com isso, conseguimos manipular as informações que queremos exibir de forma mais eficiente e com menos código.

Outra diferença é que não temos a propriedade `crossAxisCount` e é obigratório colocar um `gridDelegade` para especificar quantas colunas precisamos dentro do nosso layout. Apesar de ser uma forma um pouco mais complexa de criar um *grid*, essa forma também provém mais liberdade para que o grid fique exatamente da forma que desejamos.

Caso queira saber mais, você também pode conferir na própria documentação do Flutter (https://api.flutter.dev/flutter/widgets/GridView-class.html#widgets.GridView.2).